Restaurant Management System

Moukhik Misra

*Computer Science and Engineering (B. Tech), Vellore Institute of Technology*

*Vellore, India*

Email: [moukhik.misra2019@vitstudent.ac.in](mailto:moukhik.misra2019@vitstudent.ac.in)

Seetha R.

*Vellore Institute of Technology, Vellore, India*

[rseetha@vit.ac.in](mailto:rseetha@vit.ac.in)

***Abstract*-- The restaurant industry is a highly popular and fast-paced industry worldwide. This project, developed on C programming language aims to solve the problems created by such a fast paced and information heavy industry by bringing innovative and efficient solutions as well as simulating the working of a restaurant manually using a turn-based program. This system allows users to create and modify their menu by setting the name, price and the number of turns it takes to prepare an item. The program is menu-driven and has several options for both the customer as well as the restaurant manager. It is a system which allows customers to order items from either starter, main course or dessert menu digitally and this information is relayed directly to the kitchen to ensure the highest possible accuracy in the ordering process. Each customer is assigned to a free table and has a unique table id. On the manager side, this program allows the manager to view and modify the menu items as well as to view the customers at any table. It also allows the manager to view the food items which are currently being prepared in the kitchen as well as the number of turns left for them to get prepared. Bills are generated for a customer based on their orders and is displayed to the customer as they leave the restaurant. Quick sort and stack data structure are used to show the preparation of food going on in the kitchen at a particular turn as well as to give us the orders which have been prepared in a particular turn. The program also uses several user-defined data types and arrays of such data types to store information of customers and the orders placed by them. The food menu is stored in the form of a linked list.**

***Keywords*--Restaurant Management System, Linked List, Stack , User-Defined Data Type, Quicksort, Arrays, Menu-driven, Robust, Efficient, Billing, Time-Complexity, Customers, Manager, Turns.**

I. INTRODUCTION

The restaurant industry is a largely manual industry. It relies heavily on its staff to perform optimally. This approach has been used for a long time. However, the world today is gravitating towards digital platforms. In this technology driven era, the restaurant industry needs to find new ways to become more efficient and effective. This requires a transformation of the current manual system to a robust and digital system which can both simplify the management of the restaurants during rush hour as well as enhance the customer experience.

With this program, managers can view different facets of their restaurant in a way which minimizes the time required to perform any task. Customers are automatically assigned to a free table thereby making the wait process faster. Customers also have a user-friendly interface which makes ordering food simple and satisfying as well as being highly accurate and transparent on waiting times. Kitchen staff are directly notified of incoming orders from particular tables and can hence begin work right away. The billing is also automatically done and is relayed to the customer as the leave the restaurant. It also refreshes automatically. This makes service efficient and enhances customer satisfaction. This program simulates the working of a restaurant using a turn-based system where time is replaced by turns. Both manager and customer are given a plethora of options to choose from making it easier for both of them.

II. EXISTING TRADITIONAL SYSTEM

A manual ordering system is still used to take and place orders at most restaurants where waiters are used to take orders and relay these orders to the kitchen staff. This system is prone to errors in communication and also can lead to delays in service. Billing is also mostly done manually by entering the items into a billing program or by paper. This can lead to error in calculation and is also very time intensive.

The manager of the restaurant also has to manually check on what orders are being prepared in the kitchen. The lack of proper details of the workflow can lead to errors in communication and timely service of customers which can affect the customer experience and restaurant reputations. Updating the menu is a manual task.

Customers have to wait for a while to check if a seat is available in the restaurant or not. Ordering is not efficient as it is requires a waiter to be free and is prone to communication errors. Customers cannot place orders at their own leisure.

The lack of a solid framework which unites all these facets integral to the efficient and profitable functioning of a restaurant is a huge disadvantage for the traditional system of running a restaurant.

III. PROPOSED SYETEM

This project gives very efficient solutions the problems that exist in traditional restaurants. Customers do not have to wait for a person to tell them if a table is free or not, all this is done instantly. Customers also have the freedom to order at their own pace as well as the convenience of selecting the item in the menu and knowing that it has been relayed to the kitchen staff instantly. This saves a lot of time and resources for the restaurant and enhances customer satisfaction.

Restaurant managers, with this application, have plenty of functions at their disposal. They can create and modify the menu with ease. They are also able to view the customers currently sitting in the restaurant as well as easily view the workflow in the starter, main course, dessert section of the kitchen with the options provided by this application.

This application is a strong framework which streamlines operations for a restaurant and increases accessibility and satisfaction of the customer thereby improving over the existing traditional system of restaurant management.

IV. EXISTING WORK

*A. Algorithms*

*1) Quicksort*

Quick sort is a highly efficient sorting algorithm which uses divide and conquer technique to partition arrays into smaller arrays. A large array is partitioned into two arrays one of which holds values smaller than the specified value, say pivot, based on which the partition is made and another array holds values greater than the pivot value. Quicksort partitions an array and then calls itself recursively twice to sort the two-resulting subarray. Time complexity of quicksort for the average case is O(n logn) and Space Complexity for quicksort is O(logn) thus making it very useful for large datasets. In this project quick sort is used to sort the items in the kitchen stack in descending order of turns left to make then easier to remove.

Divide and Conquer: In divide and conquer approach, the problem in hand, is divided into smaller sub-problems and then each problem is solved independently. When we keep on dividing the subproblems into even smaller sub-problems, we may eventually reach a stage where no more division is possible. The sub-problems are then solved and merged to obtain the solution of the original problem. Steps involved are divide/break, conquer/solve, merge/combine.

*B. Data Structures*

*1) Stacks*

A stack is an Abstract Data Type (ADT), commonly used in most programming languages. A stack allows operations (insertion, deletion) on one end only. This feature makes it LIFO data structure. LIFO stands for Last-in-first-out. Here, the element which is placed (inserted or added) last, is accessed first. In stack terminology, insertion operation is called PUSH operation and deletion operation is called POP operation. In this application, stacks have been used to store the operations of the kitchen

*2) Linked List*

A linked list is a sequence of data structures (node), which are connected via links. Linked List is a sequence of nodes which contains one or many items. Each node contains a connection to another node. Linked list is the second most-used data structure after array. In this project linked lists have been used to store the menu.

*3) Arrays*

An array is a data structure that can store a fixed-size sequential collection of elements of the same type. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.

V. PROPOSED WORK

In this project we have used several user defined data types to store information about customers such as their name, phone number as well as defined a structure which hold the billing details of one particular customer. This allows the application to generate a bill for a specific customer with ease. The structure holding the billing details is nested inside the structure which contains customer details, hence tailoring the bill to that particular customer. An array is defined of user defined type that stores the name of the item and its price. The bill is updated when the customer orders another item and is refreshed when the customer leaves.

An array of user defined data type is used to store the details of all the customers. Each customer is assigned to a table with a particular table id. This id is the link which is used to determine which table has placed the order and which bill is to be updated. This data type also contains a key called avail which is 0 if table is free and 1 if table is occupied. If a new customer comes, they will be assigned to the table with the smallest table id with avail key=0. If no tables are free then customer is given that message.

A linked list has been used to store the details of the items in the food menu. The food menu has been divided into three separate menus, the starter menu, the main course menu and the dessert menu. Each of these menus is a separate linked list with nodes which contain the details of the particular item such as name, price and number of turns required.

The kitchen has been divided into three parts namely, the starter kitchen, the main course kitchen and the dessert kitchen. Each of these kitchens corresponds to a stack which stores the information of the food item ordered by the different customers. This stack is of user defined data type and it contains name, number of turns and table id of the given order. The table id links the item back to the person who has ordered the food.

This application is a simulation of a restaurant. It works on a turn-based system as a unit of time. When the user ends the turn it means that a particular amount of time has passed. This allows us to simulate a restaurant situation. At the end of each turn the turns required to prepare the items in the kitchen stacks are decremented by 1.

Quicksort has been used to sort the stack of food items in descending order based on the number of turns left to prepare it. Quick sort has been used as it is the fastest sorting algorithm available i.e. it has the best time complexity among sorting algorithms with an average time complexity O(n logn) and average space complexity O(log n). Quick sort allows the manager to view the workflow in each of the kitchens which saves time for service and other tasks. There is no confusion in this case about the time required to prepare the food. Quicksort has been used over merge sort as quicksort has a better space complexity (O(log n)) compared to merge sort ( O(n) ) which requires auxiliary space to store the sub arrays when it runs. Quicksort is clearly the more efficient algorithm when it comes to sorting arrays. Quicksort in association with the pop() function of a stack also allows for faster deletion of items from the stack. Items are deleted from the stack of items being prepared when turns left of that item becomes 0. Usually time complexity of deleting unsorted arrays is O(n) but as the arrays have already been sorted beforehand the time complexity is now reduced to just O(1) which increases the efficiency. The drawbacks of quicksort have also been rectified by this method. The worst-case time complexity of quick sort is O(n2) which occurs when quicksort is used on a sorted array which can occur in the program. However as new orders are pushed onto the sorted stack every turn and these orders require different amounts of time to be prepared which randomises the stack again. Hence the chances of encountering the worst-case are significantly lowered.

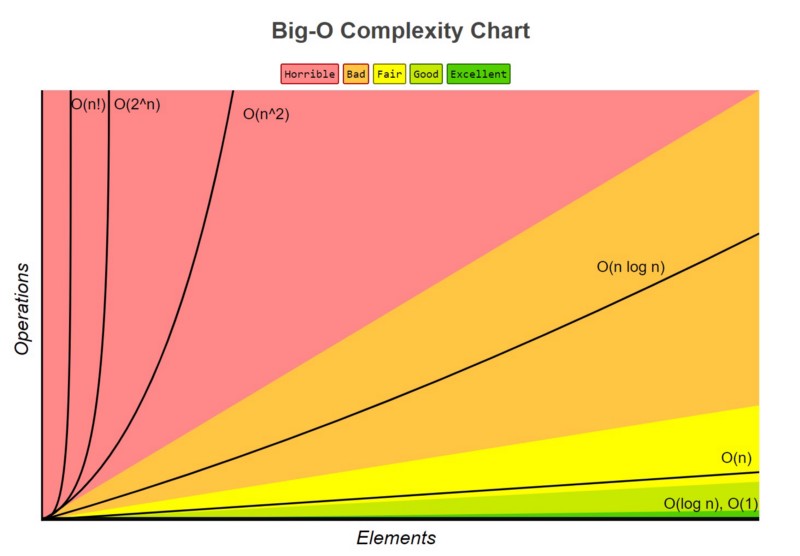


Fig. 1. Table showing different time complexities as a graph4

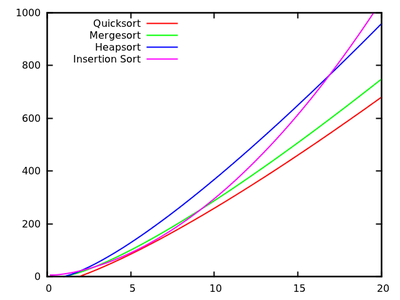


Fig. 2. A closer analysis of sorting algorithms time complexity and running times.

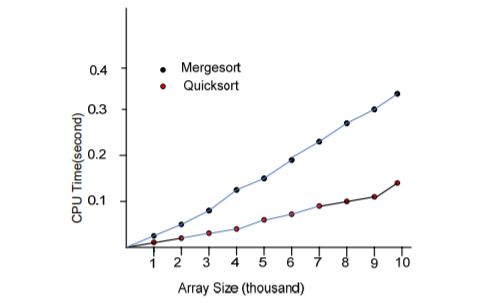


Fig. 3. Shows the runtime of Quicksort vs Mergesort

VI. WORKING

This section has been divided into parts which represent the different part of the application.

*A. Creating and updating the menu*

Define struct starter { // a structure to hold information about food item

Define float price, char name[100], int ut (which is the turns taken), int key, int ckey;

Define struct starter next\* next

};

Define struct starter \*heads,\*tails=NULL

Define function void add\_starter()

{

//This function is used to create a linked list of all the starter items. A new node is defined using malloc function and this node is added to the end of the linked list

}

Define void disp\_starter()

{

//This function is used to traverse the list and display the name and price of the items on the menu

}

Define void del\_starter(int pos)

{

Define struct starter \*temp=heads,\*temp2=NULL,\*del=NULL

If heads==NULL then linked list is empty

Else

Delete the node at the position which has been passed down by the function and free that node using free(del) // del being the node that is to be deleted

After deleting the item update the key of all the items on the list.

}

These same steps are followed to create and modify both the main course and dessert menus. Different structure and function names are used but their functions are identical to these functions.

*B. Kitchen food preparation order*

Define struct item\_stack{// structure which stores the details of the items being prepared

Define char name[100], int tid (which is table id) , int turns

};

Define struct item\_stack start[100] // We have set maximum number of starters that can be prepared at one time to be =100. This is the starter stack.

Define sttop= -1; // This value represents the value of the top of the stack

Define void starter\_list\_add (int cy, int table\_id, int turn)

{

If (sttop is 99) then PRINT “The starter kitchen is full”

Else: Push a starter with the name, table\_id and number of turns onto the stack.

}

Define void starter\_pop() // to pop starter stack

{

This function pops all those items from the stack whose turns left are 0

Decrement sttop by 1 for every item popped

}

Define quicksort\_starter (struct item\_stack start[100],int first,int last) // applying quicksort on the stack

{

Define int i,j,pivot

If (first < last)

{

Set pivot=first, i=first, j=last

while( i<j)

{

while(start[i].turns>=start[pivot].turns&&i<last)

{ i++ }

while(start[j].turns<start[pivot].turns)

{ j-- }

If( i<j)

Swap items and all the details between start[i] and start[j]

}// End of outer while loop

Swap the items and all the details between start[j] and start[pivot]

quicksort\_starter(start,first,j-1)// recursively caling the quicksort\_starter function to sort first half of the array in descending order based on number of turns remaining

quicksort\_starter(start,j+1,last) // recursively calling the quicksort\_starter function to sort other half of array in descending order based on number of turns remaining

Define void view\_starter() // function to view the starters being prepared in the kitchen

{

Display the starters in the stack

}

These same steps are followed to create and modify both the main course and dessert menus. Same structure is used but different function names are used but their functions are identical to these functions.

Define void decrement() // to decrement number of turns remaining

{

Quicksort all three stacks using their respective functions.

This function is used to decrement the value of turns remaining of all items being prepared in the kitchen (in the three stacks of starter, main course and dessert).

}

*C. Customer Functions*

Define struct customer // structure which stores the details of the customer

{

Define char name[100], int phone;

Define struct bill (inside struct customer)

{

Define char name\_item[100], float price;

};

Define struct bill bill\_cust[100]; // nested structures which contains the bill of the customer

int avail; // this is the key which shows whether a table is free or not (0=free, 1= occupied)

}

Define struct customer table[20] // 20 tables in total in this restaurant

Define void\_initial()

{

This function is used to initialise all the values of array table.

Bill price is initialised to -1

}

Define void create\_customer(int table\_id)

{

Enter the details of the customer such as name and phone number and set avail to 1 and assigns to table\_id

}

Define int check\_freetable()

{

This function checks which table is available and return the table id of the smallest value free table.

If no free table is available, it returns 21;// max number of tables is 20

}

Define void display\_customers()

{

Display the name of the customer along with the table number

}

Define void select\_starter(int key)

{

This function allows the customer at passed table id to select an order from the starter menu. The selected item is added to the starter stack.The function also allows for multiple orders of starters. The selected starter is also added to the bill of that customer.

}

Similary select\_main(int key) and select\_dessert(int key) are two other functions that allows a customer to select from main course and dessert menu respectively

Define void generate\_bill(int table\_id)

{

This function generates the bill for a customer at table <table\_id> and it adds the price of all ordered items (whose price is not -1) using the bill\_cust[100] array. Gives the sum and adds 18% GST tot his price and displays name, phone number of customer and displays all the details.

}

Define void renew\_bill(int table\_id)

{

Sets the price of all 100 elements of bill\_cust array for that particular customer to -1

}

*D. Menu*

Define void customer\_menu()

{

Runs a switch case inside do-while to give options and calls required functions till exit choice is chosen.

1.Add a customer

2.Let a customer leave the restaurant

3.Place Order

4.End turn // ending turn decrements, quicksorts and pops the stacks

5.Go to previous menu

Default: “Enter a valid choice”

}

Define void view\_kitchen()

{

Runs a switch case inside do-while to give options and calls the required functions till exit choice is chosen.

1.Starter Kitchen Preparations

2.Main Course Kitchen Preparations

3.Dessert Kitchen Preparations

4.End turn // ending turn decrements, quicksorts and pops the stacks

5.Go to previous menu

Default: Enter a valid choice”

}

Define void manager\_menu()

{

Runs a switch case inside do-while to give options and call the required functions till exit choice is chosen

1.Modify Menu

Choice 1 runs a nested switch case inside a do-while loop with options:

i. Add to Menu

Choice i runs a nested switch case inside a do-while loop with options:

a. Add to starter menu

b. Add to main course menu

c. Add to dessert menu

4.Go to previous menu

ii. Delete from Menu

a. Delete from starter menu

b. Delete from main course menu

c. Delete from dessert menu

d. Go to previous menu

iii. Go to previous menu

Default: “Enter a valid choice”

2.View Menu

3.View Customers

4.Kitchen Queues

5.End turn

6.Return to previous menu

Default: “Enter a valid choice”

}

Define void menu() // this is the main menu

{

Runs a switch case inside do-while to give options and call the required functions till exit choice is chosen.

1.Create or modify menu

Choice 1 runs a nested switch case inside a do-while loop with options:

i. Add to Menu

Choice i runs a nested switch case inside a do-while loop with options:

a. Add to starter menu

b. Add to main course menu

c. Add to dessert menu

d. Go to previous menu

ii. Delete from Menu

a. Delete from starter menu

b. Delete from main course menu

c. Delete from dessert menu

d. Go to previous menu

iii. Go to previous menu

Default: “Enter a valid choice”

2.Customers

3.Restaurant Management

4.End Turn // decrements, quicksorts and pops the stacks

5.Exit

Default: “Enter a valid choice”

}

Define void main() //main() function of the program

{

menu() // this function runs the entire program

}

This is the end of the program

VII. Output

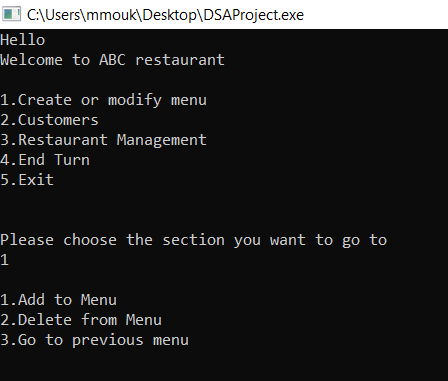


Fig. 4. Opening menu of the program

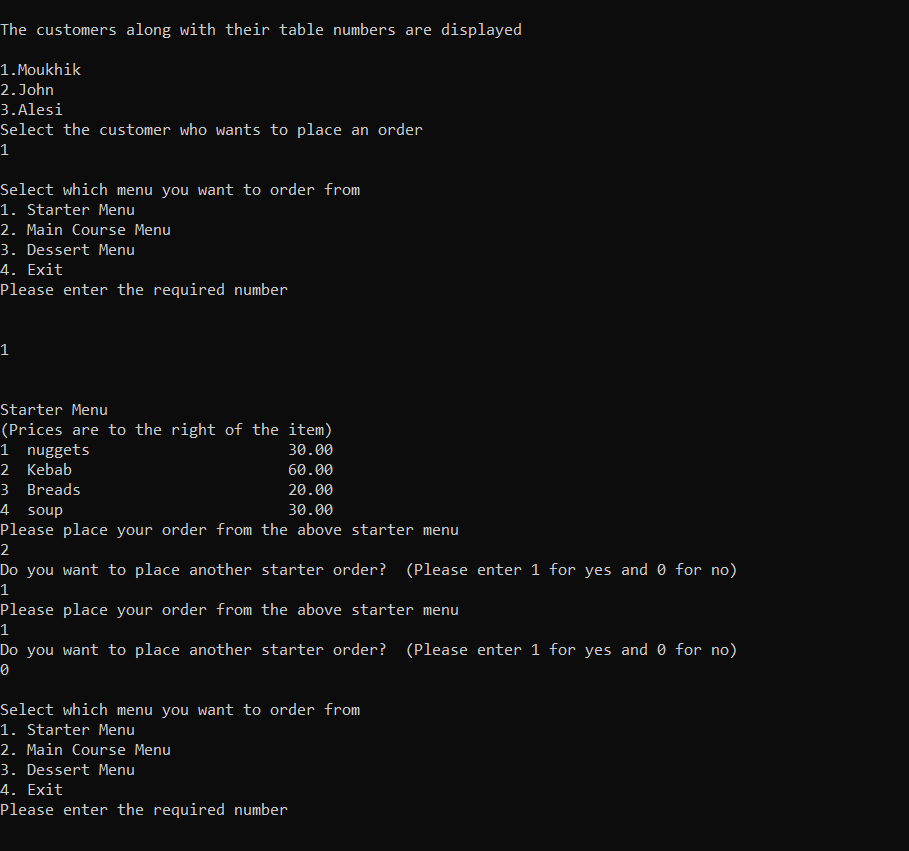


Fig. 5. The ordering interface

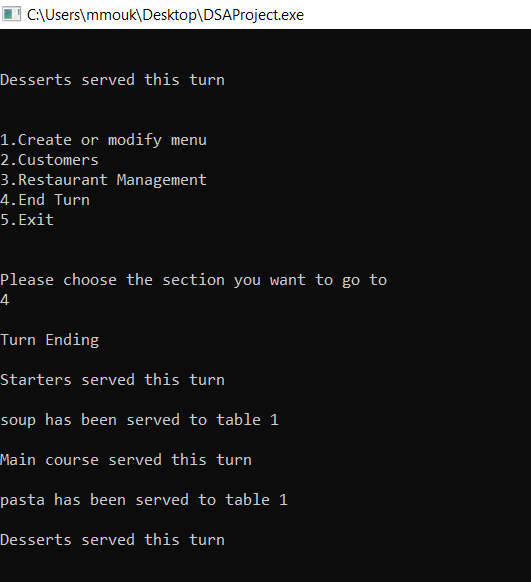


Fig. 6. Illustrates the turn system of the program

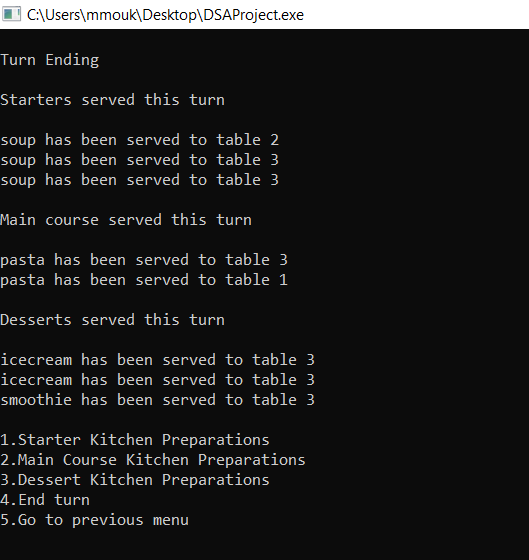


Fig. 7. Orders being delivered to their respective tables

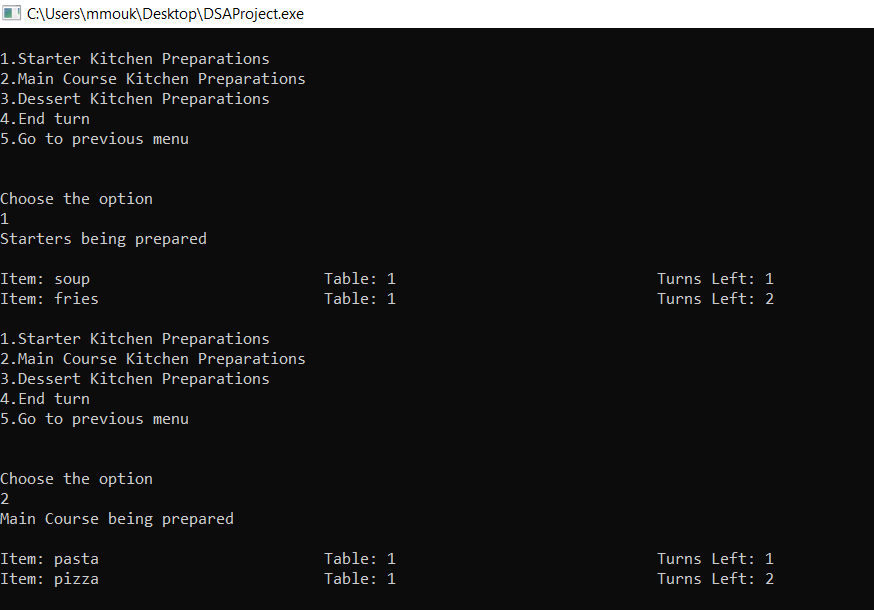


Fig. 8. Preparations going on in the kitchen in ascending order of turns left

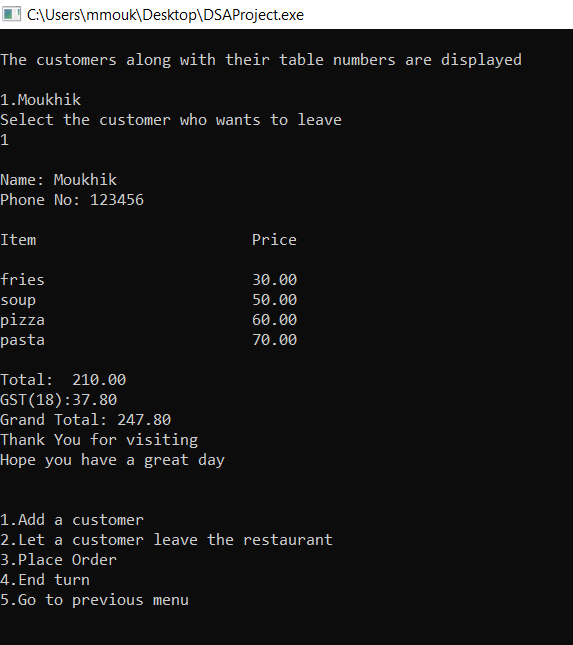


Fig. 9. Bill of a customer

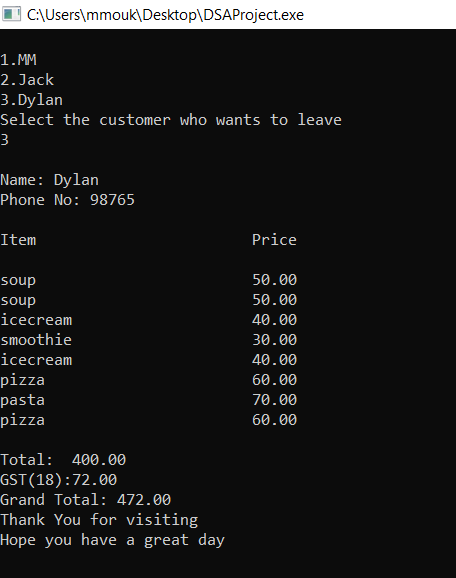


Fig. 10. Bill of a customer named Dylan

VIII. ADVANTAGES

1. The shortcoming of quicksort is overcome in most cases when the application is in operation.

2. The combination of quicksort and pop() is an efficient method of deletion in this application and it serves several other purposes.

3. This application streamlines a lot of the tasks that are done in managing a restaurant.

4. Great ease of access features are available which are beneficial to customer satisfaction.

5. Simple and clean interface is useful for the customer.

6. Multiple functions are provided for the manager which makes their job easier.

7. A quick and efficient billing system is provided.

8. Kitchen work orders are well managed, and details are well elucidated.

IX. CONCLUSION

As we move further and further into the digital age, it is apparent that all industry’s need to become more digital reliant. This applies to the restaurant industry as well. This solution is extremely time-efficient and is a right step into the direction for restaurants. It uses simple existing data structures and algorithms to create unique and efficient combinations which satisfy the needs of the restaurants. Quicksort and stacks of user defined data type have been together to come up with a solution which negates the shortcoming of quicksort. Customer data is stored and each customer is referred to by their table id which is a unique code for each table. This simplifies the ordering and billing process. This application is a means by which restaurant service can be streamlined and it reduces human error by a large degree. A simple and clean interface has been provided to improve ease of access. This program also incorporates a billing system that is highly effective, generated customer by customer and automatically refreshing itself if a customer leaves. It provides several key options to a manager such as viewing the work going on in the kitchen. All these options make the experience more streamlined and efficient for both customer and manager. This application also simulates the working of a restaurant making the program engaging and enjoyable for the user.

X. LINK TO VIDEO EXPLANATION

ACKNOWLEDGEMENT

The author wishes to acknowledge Seetha R. for giving the broad topic on which this project is based. The author also wishes to acknowledge all those who have supported him throughout the development of this project.

REFERENCES

[1] Hoare C. A. R (1961), Algorithm 63, Partition 63; Partition 64, Quicksort; *Communications of the ACM,* Vol 4. P.321

[2] C.A.R. Hoare: Quicksort. Computer Journal, Vol. 5, 1, 10-15 (1962)

[3] Sedgewick, Algorithms in C++, ISBN 0-201- 51059-6, Addison-Wesley, 1992

[4] Song Qin, Quicksort algorithm

[5] <https://www.geeksforgeeks.org/structures-c/>

[6]<https://www.javatpoint.com/array-of-structures-in-c#:~:text=Array%20of%20Structures%20in%20C,entities%20of%20different%20data%20types.>

[7] Karuna, Garima Gupta, “Dynamic Implementation using Linked List” IJERMT- September 2014, Volume 1, Issue-5, ISSN: 2348-4039

[8] V. Swapna, M. Firdouse Ali Khan, “Design and Implementation of Ordering System for Restaurants”, in International Journal of Engineering Research & Technology (IJERT), Vol. 1, Issue 10, December- 2012.

*A. Google Drive*

<https://drive.google.com/file/d/1COGXWdfHriejqVKIYAH72IEcrmUlYqWF/view?usp=sharing>

*B. YouTube*

https://youtu.be/H3-iFV8hYPc